National College of Ireland

Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing

MSc Research Project
Masters in Cloud Computing

Tamaraebi Besife Pibowei
Student ID: x20217871

School of Computing
National College of Ireland

Supervisor: **Sean Heeney**

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Tamaraebi Besife Pibowei
Student ID:	x20217871
Programme:	Master of Science in Cloud Computing **Year:** 2021
Module:	……Research Project…….……
Supervisor:	Sean Heeney
Submission Due Date:	……16th December 2021…….……
Project Title:	Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing
Word Count:	7635 **Page Count:** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Tamaraebi Besife Pibowei……

Date: ……09th December 2021……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Combined Genetic Algorithm and Gradient Descent Algorithm to Optimize Server Selection in Mobile Edge Computing

Tamaraebi Besife Pibowei
x20217871

Abstract

MEC has become the new frontier in minimizing latency in data transmission for mobile devices that are limited in resources. Firstly, the constraints of an edge server in terms of processing capacity and distance that can be covered result in only a limited number of users being able to have their requests completed at the same time. Secondly, because of the varied geographical locations of user mobility pathways in MEC, edge user mobility is significantly connected to data transmission rate and influences edge server latency. Furthermore, when multiple users in an edge server covered region require the same resources at the same time, they interfere with each other and may reduce the experience of service if an effective strategy for requests distribution to different capable edge servers is not in place. Therefore, to reduce latency and computational overhead, we consider the above constraints and propose a novel approach that combines genetic algorithm (GA) and gradient descent (GD) to find an approximately solution to the edge server selection optimization problem. We validate our model by conducting experiment on the EUA dataset. The outcome of the experiments conducted demonstrates that, our model significantly outperforms the baseline approaches.

1 Introduction

Swift advancements in mobile developments and its applications in our daily business interaction activities such as natural language processing, facial recognition, internet of things and interactive online entertainment and have garnered interests from both academia and industry experts. This paradigm shift in cloud computing and mobile networks has turned computing from being managed on-site to virtually hosted hardware and software services (Zhang et al., 2012). Cloud computing is a consolidation of hardware and software components; thus, a significant amount of computing functionality is offloaded to distributed cloud frameworks such as fog and mobile edge computing (MEC) to the accessibility of end users which translates to the reduction of latency and computational limitations mobile devices and centralized cloud architecture (Yuan, Guo and Wang, 2021). While the capability of mobile devices could be considerably scaled up by such offloading events, communication latency becomes unavoidable since the remote cloud infrastructure are usually far away from the mobile users. For applications that are time-sensitive, long transmission duration can noticeably reduce the user's quality of experience (Tan et al., 2017).

MEC, as an extension of centralized computing aims to bring computing resources closer to mobile users by adding edge servers to the edge of the network, it still faces the fundamental issue of limited edge server resources. While a user might want to connect to the edge servers

to fulfil certain service requests made, the edge servers might not have enough computing resources to satisfy the requests made (Zhang et al., 2021). This problem is further reinforced with users' mobility and multiple users request, therefore, selecting the optimal edge server from the solution space. Each edge server is usually delegated to provide coverage to a dedicated geo-location and users can connect to them when they are withing the range of coverage. Given a user's mobility, when a user migrates from the coverage of the original edge server with uncompleted tasks, the user's connection to another edge server needs to be established where the computation task will be offloaded to guarantee service continuity (Xenakis et al., 2013). This demonstrates that, if a user continues to utilize the initial edge server for data transmission through other nodes, more computational resources will inevitably be consumed, leading to increase in latency as a result of other nodes' resources being occupied. Therefore, the best policy for selecting the optimal edge server for service migration should be implemented, the policy should include a decision for if a service should be migrated and where it should be migrated to. Between the user and the MEC, there is a bater between the cost of service migration and cost of transmission (communication latency and network overhead respectively). Determining the best selection is difficult because of the unpredictability in user mobility as well as the likely non-linearity of migration and transmission costs (Wang et al., 2015).

To mitigate these issues, the edge server a user would migrate to should be selected in advance. As mobile users move, several edge servers must be selected in order for the user to receive resources. To improve the quality of experience, the servers with the widest coverage and more resources should be selected (Yin et al., 2016). The figure

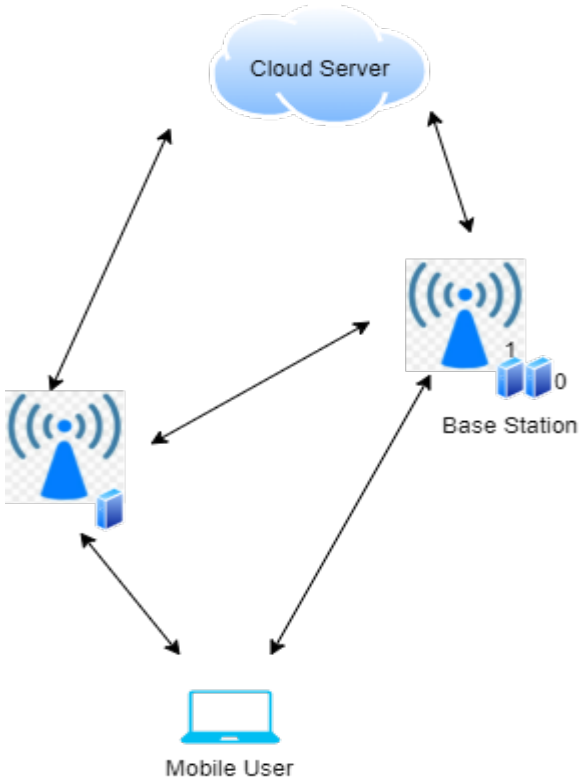

Figure 1: System Model Architecture.

In this paper, we look at how to choose the optimal edge servers for mobile users in order to reduce the user's overhead. The overhead of mobile users will be influenced by elements frequency of service migration, scarce edge server resources, energy constraint of mobile devices (Wang et al., 2018; Shah-Mansouri and Wong, 2018; Tianze et al., 2017). In this research, we present the combined genetic algorithm and simulated annealing algorithm for edge Server Selection approach to overcome the aforementioned challenges., the proposed approach can, in polynomial time, pick edge servers for the mobile user. The following are the key contributions:

1. We simulate the job completion probability of edge servers and examine the user's computation cost in non-migration scenario of edge server selection.
2. In order to reduce user overhead in terms of time delay in mobile environments, we present the Combined genetic gradient descent (CGGD) algorithm, which picks the best edge server in a large solution space for users with known paths.
3. To evaluate the performance of our CGGD framework, we perform comprehensive simulations based on a real-world data. The outcome of the experiments demonstrates that CGGD can effectively minimize the user's overhead.

The rest of the paper is organized as follows: in section II we discuss the most recent research done and their approach. In section III, we give details of our approach, the experiment design, details of the algorithm and implementation. Then, in section IV, we discuss the experiment setup and outcome obtained. We then discuss the result obtained and give a summation of our research.

In this section you introduce the topic of the paper, motivate why it needs to be studied (appropriate citations are best for this), presents the research question(s) and research objectives, and/or hypothesis or hypotheses. Briefly summarise the contribution to the scientific literature your work entails. Finish this section by outlining the structure of the report.

2 Related Work

A lot of solutions to edge server selection problems have been proffered by many scholars (Zhang et al., 2019) introduced a Multi-user Edge server Selection method based on Particle swarm optimization (MESP). The model selects edge servers ahead of time for mobile users within polynomial time. Although the approach can effectively reduce the total waiting period, however, the approach only performs well when the path of the mobile users is known. (Zou et al., 2021) proposed a genetic algorithm-based model, GASISMEC. The model is built around three factors – complexity of crossover, response time-aware mutation operation and fitness calculation. The model performs service instance selection in polynomial time by firstly, generating a random group of allocation strategies and evaluating the response time during which the response time-aware mutation operation is initiated. In their study, (X. Chen et al., 2018) proposed a deep Q-network-based offloading policy modelled as a Markov decision process to reduce the long-term cost an offloading decision between a mobile user and multiple base stations. The model proposes to learn the best policy without having previous knowledge

of the dynamic data such as the quality of the channel. However, the model does not account for multiple users and the numerical calculation shows 56% increase in performance. (Zhao et al., 2019) proposed A cross-edge computation offloading (CCO) framework based on Lyapunov optimization algorithm (Chen, Wang and Li, 2019) without prior knowledge to determine the edge server selection for partitionable applications. The framework proposed can achieve asymptotical optimality with battery capacity of mobile devices. (Alchalabi et al., 2021) proposed a Quadruple Q-Learning model that minimizes the variance of latency by using geo-distance between the users and the edge server. The model uses Reinforcement Learning (RL) method to reduce the action space which emphasizes fairness in edge server selection. The framework adopted a modified version of Action Elimination Network (AEN) for linear vector rather than neural network for a tabular case.

A MEC server selection policy which is reliant on the MEC server's resources and link parameters was introduced by (Dilanka et al., 2021). The Mobile Edge Orchestrator determines which server is selected based on the MEC server infrastructure, linked bandwidth, and distance between end-user and MEC server. The requests are handled based on a queue structure, however, priority on more important requests is not considered in the execution order.

An online secretary framework is proposed by (Lee, Saad and Bennis, 2017) to minimize the maximum computational latency for a fog network. In this framework, the initial fog node cooperates with other neighbouring fog nodes and the cloud data centre to build a fog network and distribute tasks. In the midst of uncertainty, any fog node can dynamically create a fog-network by choosing the most appropriate set of fog node neighbours where tasks will be distributed among the fog nodes and the cloud without prior knowledge of future incoming fog nodes or their performance capacity. Similarly, (Shah-Mansouri and Wong, 2018) proposed a model to efficiently allocate the sparse processing power of fog nodes in IoT environment where each user can strive to improve its own quality of experience in a computation offloading game. Each user task can be executed locally or offloaded to computing servers which comprises a set of fog nodes and cloud servers. Each user is greedy and therefore, the fog nodes cannot emphasize applications of an IoT user over other IoT users. In contrast, they propose a Nash Equilibrium (NE) algorithm to obtain the equilibrium. To reduce the time complexity of achieving the Nash Equilibrium, they proposed the adoption of near–optimal resource allocation algorithm. (Chen, Liang and Dong, 2017) proposed a three-step algorithm to collectively optimize the offloading decisions of all users' tasks while allocating computation and communication resources in a bid to decrease the total cost of power, computation, and latency for all users in a mobile cloud environment. The steps which are semidefinite relaxation (SDR), alternating optimization (AO), and sequential tuning (ST) can optimize the process independent of one another. However, they opined that when combined, the optimization is more efficient. A computing task caching policy (Edge-CoCaCo) was introduced by (M. Chen et al., 2018) which uses computing task caching placement and task offloading decision optimization to curb latency in task processing and improve users' quality of experience. The framework proposed a combined optimization of computation, caching and communication on the edge cloud. Popular tasks, are cached on the edge cloud and is not required to be offloaded by the mobile user. Local processing is done for tasks that have large data size but require a lot of data transmissions and small computing resources. However, for tasks with fewer data

transmissions requirements but computation-demanding resources, such tasks can be offloaded to the edge cloud for processing using a branch and bound algorithm.

To address service interruption caused by vehicle mobility and the inadequate edge coverage, (Tang et al., 2021) proposed a pre-allocation algorithm for vehicle tasks. Their study demonstrates how network selection can be adaptive and efficient by breaking huge tasks into smaller sets and offloading computational tasks to the newly added cells to the edge server in a vehicular edge computing environment. A technique that shares subtle similarities in computation offloading proposed by (M. Chen et al., 2018).

A heuristic approach was adopted by (Guidara et al., 2015) to choose the best possible combination of services to deal with the temporal properties that impact QoS. The proposed approach is based on clustering achieved via k-means algorithm, constraints decomposition techniques and local selection.

Given the time-consuming characteristics of randomly searching through a large pool of possible solutions, (Kasi et al., 2021) proposed a heuristic approach using genetic and hill-climbing combined with annealing algorithms to find the best solution against the multi-objective constraint optimization problem in the least number of possible solutions available for edge server placement.

(Zeng et al., 2019) proposed greedy techniques that routinely chooses nodes that can connect as many other nodes as possible under multiple constraints that pertains to the delay, degree and cluster size and also to lessen the capacity constraint of the edge servers in a wide metropolitan area network (WMAN). It also aimed at reducing the number of edge nodes on the network given that too many nodes in a cluster decreases the quality of service while using an annealing algorithm to globally optimize the model.

(Wu et al., 2019) took a heuristic approach to properly transmit service requests to edge and cloud servers in order to enhance the quality of service and minimize the latency in service invocations in MEC systems. The heuristic model combined genetic and annealing algorithm in mobile edge computing (GAMEC) for efficient service selection.

(Qian et al., 2013) introduced CSS for automatic selection of cloud infrastructure and proposed a stepwise application placement algorithm to address scalability constraints. Similarly, (Soltani, Martin and Elgazzar, 2018) developed a framework to automate the selection of the most suitable requirements and preferences for an application's features in an IaaS for service deployment. The model further addressed the issue of total service cost by consolidation to improve resource utilization.

(Deng et al., 2015) proffered a genetic algorithm (GA) based offloading technique to invoke multiple mobile services in workflows and makes decision on whether the services of a workflow should be offloaded to the cloud server to prioritize the optimization of execution time and energy consumption.

The network and computational capabilities of edge nodes in a video streaming service were appraised by using a score-based edge service scheduling algorithm proposed by (Scoca et al., 2018) to achieve the highest scoring mapping between services and resources to improve QoS delivery.

(Thiruvasagam, Chakraborty and Murthy, 2021) adopted Integer Linear Programming to proscribe a survivable mapping and latency-aware technique to reduce service provision cost in and ensure persistence against failure of Virtual Network Functions (VNF) in MEC cloud facility.

Authors / Year	Methodology	Objective
Lee, Saad and Bennis, 2017	Online secretary algorithm	Computational latency reduction for a fog network
Zhang et al. 2019	Particle Swarm Optimization (MESP).	Latency reduction in edge server selection
Zou et al., 2021	Genetic Algorithm (GASISMEC)	Latency reduction in service instance selection
X. Chen et al., 2018	Deep Reinforcement Learning	Latency reduction in task offloading decision
Zhao et al., 2019	Cross-edge Computation Offloading (CCO)	Latency reduction in edge server selection
Alchalabi et al., 2021	Deep Reinforcement Learning	Latency reduction in edge server selection
Dilanka et al., 2021	Mobile Edge Orchestrator (MEO)	Optimal edge server selection
Shah-Mansouri and Wong, 2018	Nash Equilibrium	Optimal task offloading
Chen, Liang and Dong, 2017	SDR-AO-ST algorithm	Optimization of power, computation, and latency in task offloading
M. Chen et al., 2018	Edge CoCaCo	Optimization of task offloading decision and task caching placement
Tang et al., 2021	Adaptive task offloading algorithm	Optimization of task offloading in dynamic environment
Guidara et al., 2015	K-means algorithm	Optimization of service selection
Kasi et al., 2021	Combined genetic algorithm and annealing algorithm	Optimization of edge server placement
Zeng et al., 2019	Greedy algorithm	Optimization of edge server placement
Wu et al., 2019	Combined genetic algorithm and simulated annealing algorithm (GAMEC)	Latency reduction in service invocation
Qian et al., 2013	CSS	Effective cloud infrastructure selection
Soltani, Martin and Elgazzar, 2018	Case-based reasoning and Multi-criteria Decision Making (MCDM)	Automated cloud infrastructure selection
Deng et al., 2015	Genetic Algorithm	Optimization of task offloading decision
Scoca et al., 2018	Score-based edge service scheduling algorithm	Optimization of task scheduling decision
Thiruvasagam, Chakraborty and Murthy, 2021	Integer Linear Programming	Computational cost reduction of virtual network function
Chen, Wang and Li, 2019	Lyaponuv Optimization	Optimization of task offloading for energy-efficient MEC

Table 1: Summary of Literature Review

3 Research Methodology

In this section, we assume that the load of each edge server fits a normal distribution over a period of time and analyse the task completion probability of edge servers. Then, we compute the overhead of completion task in cases of a non-migration scenario. Finally, we can obtain the user's total overhead.

Computation Probability Model: In this model, as the edge server provides resource to the user, the server resource is reduced correspondingly. Because of the limited resources of the edge server, there is little to no guarantee that the edge servers connected by the user can provide the resources required by the user. Therefore, it is necessary to measure how probable the task will be completed successfully, denoted by *p*. Assuming that the load of each edge server fits a normal distribution over a period of time, we set the probability density function of the edge server load as

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2}} + b$$

In this equation, x denotes the load of the edge server whose classification domain is [0, 2μμ], while f(x) is the probability when the load is x. Further, the value of μμ is equal to half of the total resources of the edge server, and we define b as a constant, that is related to the total amount of resources.

$$b = \frac{\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} + \int_{2\mu}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{2\mu}$$

If the limited resources required by the user are k, the probability that the edge server can satisfy the resource required by the mobile user equals is

$$p = \int_{0}^{2\mu-k} f(x)$$

If the connected edge server cannot satisfy the resource requested by the user, the edge server will send the request to the remote cloud to get the resource required by the mobile user. The probability is 1–p. At any time, the user can connect to the remote cloud server, and the remote cloud server always is able to provide the resource required by the mobile user.

Computation model of user's overhead: We assume that there are S edge servers in the system, where (1, …, S) denotes the set of edge servers. The mobile user can connect to these edge servers to transmit data through the wireless channel during the mobile process. When the user moves, we further denote the set of all servers available as (0, 1, …, S) where 0 represents the remote cloud server which has enough computing resources. When the edge server cannot provide enough resource to the user, because the edge server has limited

resources, the user can also get resources from the remote cloud server by directly connect to the remote cloud server.
During the movement, the task can be computed either on the edge server or the remote cloud server, so we consider it as one of the many tasks to be performed as $Ti \triangleq (Bi, Di)$. Here, Bi denotes the size of computation input data (that is, the program codes and input parameters) involving in the task Ti, and Di denotes the total number of CPU cycles required to accomplish the computation task Ti.

- **No service migration**: If a task is executed on the edge server, $d_{i.s} < R_s$ must be guaranteed when the user connects to the edge server s, where $d_{i,s}$ indicates the distance between the user handheld mobile device and the edge servers, and R_s is the coverage radius of edge server s. Assuming the mobile user can connect to only one server at any time, the time the user transfers data to the edge servers is $B_i/ri_{,s}$ for task T_i.

Here, $B_i/ri_{,s}$ indicates the data transfer rate between the user and the edge server s, which can know in advance by the user. The computing power of the edge server s is represented by Fs . Should the user maintain connection with "s" before the edge server s completes the task T_i, the execution time of task T_i on the edge server is $Di\ /Fs$. The latency that occurs when the edge server sends the computation output back to the user is neglected, due to the fact that for most applications, the size of the output in general is much smaller than the size of the input data. So, the response time of the task Ti executed on the edge server s can be given as

$$T_{i,0} = \frac{B_i}{ri,s} + \frac{D_i}{F_s}$$

We can compute the overhead in terms of processing time as

$$C_i = \lambda T \left(\frac{B_i}{ri,s} + \frac{D_i}{F_s} \right)$$

Here λT denote the weights of time consumption for the user. We assume that $\lambda T \in (0, 1)$ to prevent high latency. However, the setting of the parameters is user and application dependent, such that to minimize latency in a latency-sensitive application, the user might raise the weight of time latency. It is worth noting that edge servers cannot always satisfy user's resource requests because of their limited resources. When the remaining resources of the edge server cannot meet the requirements, the edge server needs to connect directly to the cloud server and send the request to the remote cloud. The task upload time, execution time and roundtrip bac to user's mobile device is a summation of the latency between user and cloud server.

$$T_{i,0} = \frac{B_i}{ri,o} + \frac{D_i}{F_o} + d$$

Where, ri, o and F_o represents the data transmission rate between user and cloud server and cloud server computing power respectively. In addition, the time latency overhead of task T_i processed in the cloud is expressed as

$$T_{i,0} = \lambda T\left(\frac{B_i}{ri,o} + \frac{D_i}{F_o} + d\right)$$

When the edge server lacks the required computational capacity minimum to execute the task. According to the model of task completion probability, the success rate that the edge server s completes the task is p. Thus, the overhead computation function that the edge server completes the task $T_{i,o}$ is expressed as

$$T_{i,0} = p\left(\lambda T\left(\frac{B_i}{ri,s} + \frac{D_i}{F_s}\right)\right) + (1-p)\left(\lambda T\left(\frac{B_i}{ri,o} + \frac{D_i}{F_o}\right)\right)$$

Mobile user's total overhead: When an edge server is selected, the computational overhead for completing a single task can be obtained according to computation model overhead. For this to happen, many tasks need to be completed for a user over a period of time. Due to user's mobility, the mobile user will connect to different edge servers to get resources. Next, we will go into how the user selects the edge servers in the mobile path.

The Mobile Path: A mobile path is represented by a triple $(Time, L, N)$, where:

- $Time$ is the time span during which the user is moving. It includes a set of continuous time points;
- L is the set of the user's locations corresponding to all time points in Time;
- N is a mapping function between the time points and the user's locations on the path. N: $Time \rightarrow L$.

In this definition, L divides the entire moving path into multiple segments. N represents the connection between time and location, which indicates the variable speed while a user moves. Specifically, if the time interval is small, it means the user moves with a high speed in that location; otherwise, the user moves slowly.

Selection of Edge Server: The selection of edge server during user movement is a triple (S_e, S, G), where;

- S_e represents the set of many segments, and Path = $se1 \cup se2 \cup$. (Combining all segments can compose the user's mobile path)
- M = (0, 1, …, S) represents the set of all the servers that the user can connect to
- G is a function representing the correspondence between the segment and all edge servers that cover the segment: $\forall sei \in Se, G: sei \rightarrow M$.

Since the user movement path is known in advance, according to the geographical location and the coverage radius of each edge server, we can get all potential servers that the user can connect during the moving path: (0, 1, …, S). Function G divides the moving path into many segments Se, and makes each segment be covered by the same severs. When the user moves to the position of the segment Sei, one of the s servers that cover the segment is connected to provide resource for the mobile user.

The Total Overhead: By assuming that the user's movement path, we divide it into n segments, and all servers that the user can connect are represented as a vector of length n: $< s1, s2, \dots, sn >$, $si \in \{0, 1, \dots, S\}. \forall 1 \leq i \leq n-1$ and $si \neq si+1$, if a task is not completed before the user leaves the coverage area of the edge server si, the task needs to be migrated, otherwise it will not be migrated. Assuming that there are m tasks to be completed, the user's total overhead can be obtained according to the above calculation model:
$\sum_{i=1}^{m} Ci$. Our goal is to minimize the total overhead of the mobile user: $min \sum_{i=1}^{m} Ci$.

4 Design Specification

4.1 Gradient Descent

Gradient Descent (GD) is the most often used machine learning method. It is not just used to solve optimization problems, but it is also the most prevalent approach for training any type of neural network, including deep learning. There are a few implementations of the method, but they all depend on the gradient theorem. GD is based on a convex function that alters its parameters iteratively to minimize a given function to its local minimum. These models evolve over time with the use of training data, and the cost function inside gradient descent functions as a way of measurement for assessing its correctness with each iteration of parameter changes.

4.2 Genetic Algorithm

The relationship between the GA and edge server selection problems are such that, in genetic algorithm (GA), achievable solutions are designed by select servers also known as chromosomes responding to the edge servers in each section. The chromosomes, also comprises of a set of independent servers called genes to represent the selected servers. During user movement, the locus of a gene in a chromosome expresses itself as a segment. The energy consumption, time and latency of the chromosome are implied as low if it has a high fitness due to the end user.

In GA, new candidates are generated through crossover and mutation. The overhead of the chromosome is calculated according to the fitness function where the GA algorithm is executed iteratively and approximate optimal solution is eventually achieved.

At the earlier phase, we updated the algorithm parameters such as population size, initialization, learning rate, step size, terminating condition, and mutation rate, and etcetera.

The selection process involves a simple process of reserving the superior chromosomes represented as 0's or 1's and randomly removing out a part of the inferior chromosomes while the remaining chromosomes are now identified as the parent chromosomes.

In the Cross-over process, the parent chromosomes recombine to generate new chromosomes (child chromosomes). During the process, a point is chosen at random from the chromosomes directly, after which a randomly selected parent chromosome parent loses the genes after that point and another randomly selected parent chromosome parent loses the genes before that point, hence a new child chromosome child is generated by combining the two parent chromosomes.

4.3 Combined CGGD

The ability of the genetic algorithm, as one of two commonly used heuristic algorithms lies in powerful global searching while the gradient descent is an optimization method used to find the local minima of a given function. This heuristic algorithm is called Combined Genetic Gradient Descent (CGGD) algorithms for edge server selection. It can inherit the powerful searching ability of the genetic algorithm, and the ability to avoid being trapped in saddle point during the early stages by decreasing the converge speed or increasing it to improve its efficiency. This algorithm is a combination of the two algorithms which works to achieve a derivative function from the genetic algorithm and optimize it using gradient descent. This proposed algorithm can select edge servers for the mobile user to get an approximate and optimal solution within a set amount of time from a population of several candidates. In this section, we will detail how CGGD is applied to our edge server selection process.

Firstly, we use GA to generate the fittest chromosomes from the records from the base stations. The fittest chromosome is then used as the x value which is the independent variable. The x value is then used to define the corresponding y value which is the dependent variable for the equation where anywhere the y value = 1, the distance between the user and the edge server is saved in the y column. Using the standard definition and the derivative function of the gradient descent, the slope and the intercept for the line equation is defined. Given the definition, slope and gradient is substituted into the line equation and used to retrieve the index of the fittest chromosomes based on the relationship between the x and y variables, and find the minimum of the predicted values. The index is used to retrieve the edge server from the base station.

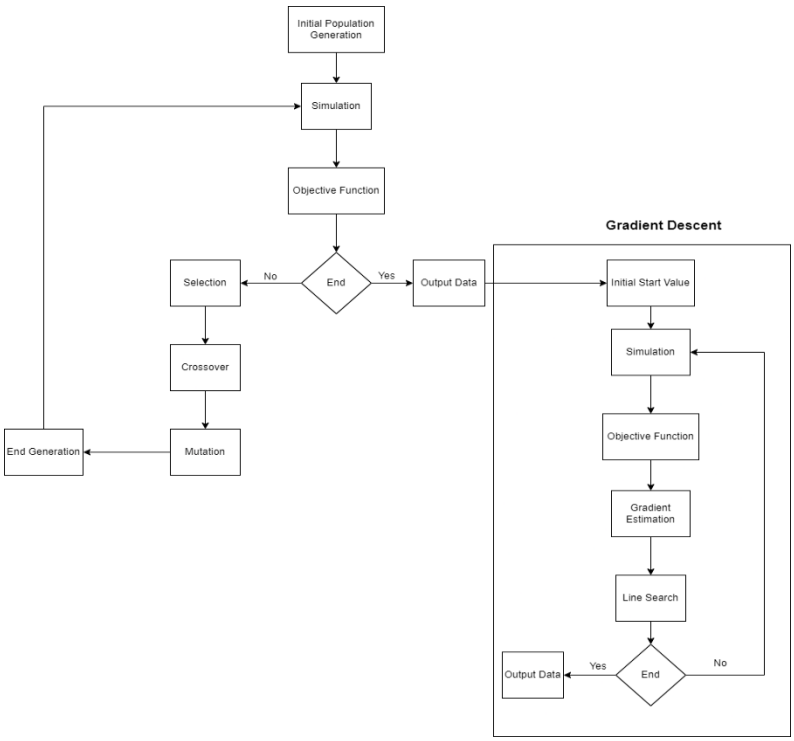

Figure 2: System Model Flowchart.

Algorithm 1: Genetic Algorithm for Server Selection in MEC

Input: SISP(U_m, List(BaseStations), P_{func}, F_{func}, S_{func}, CO_{func}, M_{func}, gl, fl)
Output: Chromosome with the highest fitness

```
population <=  Pfunc()
    for i in range(gl)
        # evaluate the fitness of every member of the population
        population <= sort(Ffunc())

        Wf <= Ffunc(population[0])
        if  Wf >= fl
            break

        next_gen <=  population[0:2]

        # Select parent for the next generation and mate selected parent to product offspring
        for j in range(len(population/2))
            # Select parents from the current population
            parents <= Sfunc(population, Ffunc)
            # Mate parent to product children
            children <= COfunc(parents[0], parents[1])
            # invoke mutation function on the children produce from mating parents
            Mfunc(children)
            # Add children to the next generation
            next_gen += [children]

        population = next_gen

        # evaluate the fitness of every member of the population
        population <= sort(Ffunc())

    SISP_GD(population[0], List(BaseStations))
```

Table 2: Genetic Algorithm

Algorithm 2: Gradient Descent Algorithm for Server Selection in MEC

Input: SISP_GD(Chromosome, List(BaseStations))
Output: base_station_instance: BaseStations

```
#create a linear relationship between Chromosome and List(BaseStation)
x <= List(Chromosome)
y <= List(compute_distance_relationship(List(BaseStation)))

define linear relationship between y and x variables
y <= mx + c

for i range(1000)
    # compute predicted y using y = mx +c
```

```
    y_pred = cur_m * x + cur_c

    compute cost
    compute dev_m
    compute dev_c
    update cur_m
    update cur_c

    if different in cost compare to previous cost is insignificant
         break out of loop

 # Predict optimal solution from List(BaseStation)
  For i, station in List(BaseStation)
      If Chromosome[i] equals 1
          Check for the BaseStation with the minimum distance value
           Store the index with the minimum value in selected_index


return BaseStation[selected_index]
```

Table 3: Gradient Descent Algorithm

5 Implementation

In this section, we carry out experiments to evaluate the performance of CGAGD and compare the results with a variety of settings in the algorithm. All experiments were conducted on a Linux operating system, fitted with intel core i7 at 3.6 GHz, 16 GB of RAM and implemented with Python v3.6.

Extensive tests are carried out on the publicly available EUA benchmark dataset (Zou et al., 2021) that is widely used in edge computing, including, to confirm the efficacy and efficiency of our technique. The EUA dataset contains data from real world sources. The dataset holds information about the user

s addresses and edge servers' locations and so on. In the experiment, we generated one dataset for the edge servers where they converge. The mobile path can be determined by plotting against longitudinal and latitudinal lines of the edge servers from the information provided in the edge server dataset. So, we can access all the base stations within a mobile user's path. We assume that the mobile user is connected to an edge server over Wi-Fi interface with an average data transmission rate of 3.01 mb/s. Given the average transmission rate over Wi-Fi, we assume that the transmission rate ranges between [2.01, 4.01] mb/s. Furthermore, we assume that the processing power of each edge server varies; ranging between [2,3] GHz. We also consider that the number of CPU of the edge server is randomly distributed between [7,18]. In this study, we do not consider the cloud server resources, given that it has ample computing resources and that there is no service migration. However, we assume that the data transmission rate from edge server to cloud server is 5.01 mb/s.

6 Evaluation

In the experiment we conducted, for each iteration, the number of users increased exponentially to a maximum of 816 users. The ratio of user requests to the edge servers is 0.05 as well as the users being drawn at random from the associated dataset's user set. To demonstrate the efficacy of the proposed algorithm, we compare the direct impact of gradient descent on genetic algorithm and against GASISMEC algorithm. The tasks to be completed are chosen at random. In order to eliminate the impact of iterations, the number of iterations for all approaches is maintained. Table 2 describes the experimental findings on EUA datasets with increased number of users.

Algorithm	**Total Average Response Time**
GA	26.61%
GASISMEC	22.10%
CGGD	15.51%

Table 2: Experimental results on EUA datasets.

Algorithm	**Average CPU Usage**
GA	35.93%
GASISMEC	87.96%
CGGD	80.83%

Table 3: Experimental results on EUA datasets.

6.1 Experiment on latency

The result demonstrates that the average response time on the EUA dataset with normal user distribution is significantly shorter than the competing baseline algorithms. It is caused by the using normal distributions of BSs in the dataset. While GA without GD performs the worst with an average response time of 25.18%, CGGD performs best when compared with another hybrid GA algorithm which records 15.51% and 22.10% respectively. Correspondingly, as the number of users grow, so does the computation requirements for processing user requests. Figure 3 below shows the average response time relative to the number of users connected to the BS.

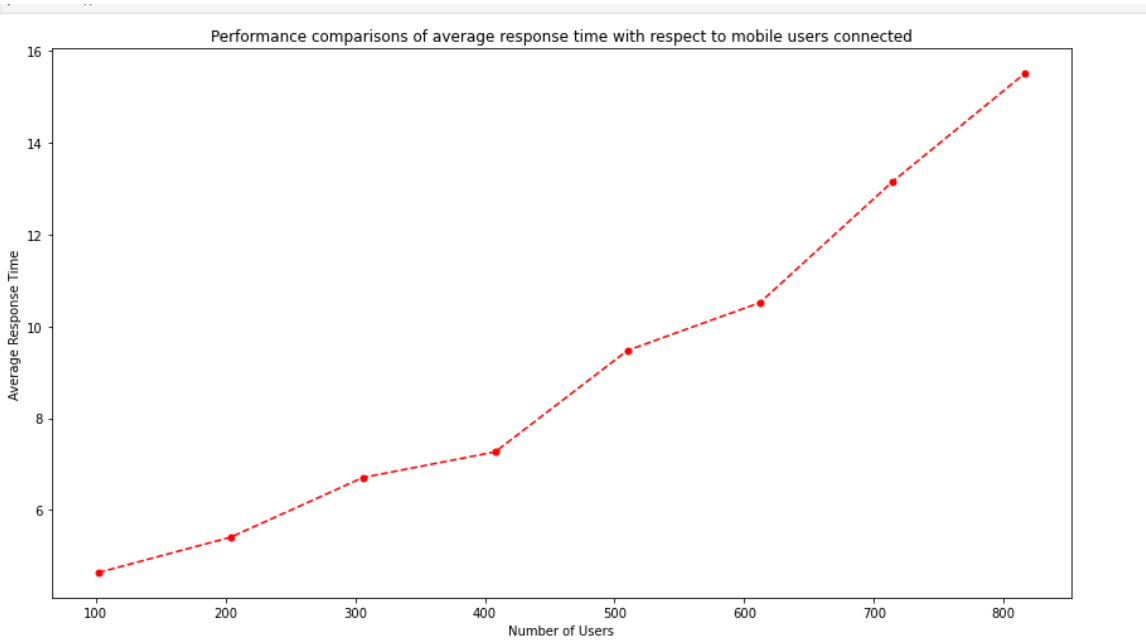

Figure 3: Performance of computational time on EUA dataset.

6.2 Experiment on total overhead

To further validate our heuristic approach to edge server selection, we calculate the total overhead. The result depicts that the average CPU usage of CGGD scales accordingly as the demand for computational resources increase. The outcome show that the 80.83% recorded by CGGD outperforms GASISMEC's 87.96%. However, GA shows to demand the least computation resource with 35.93%. Figure 4 below shows the average CPU usage relative to the number of user requests generated.

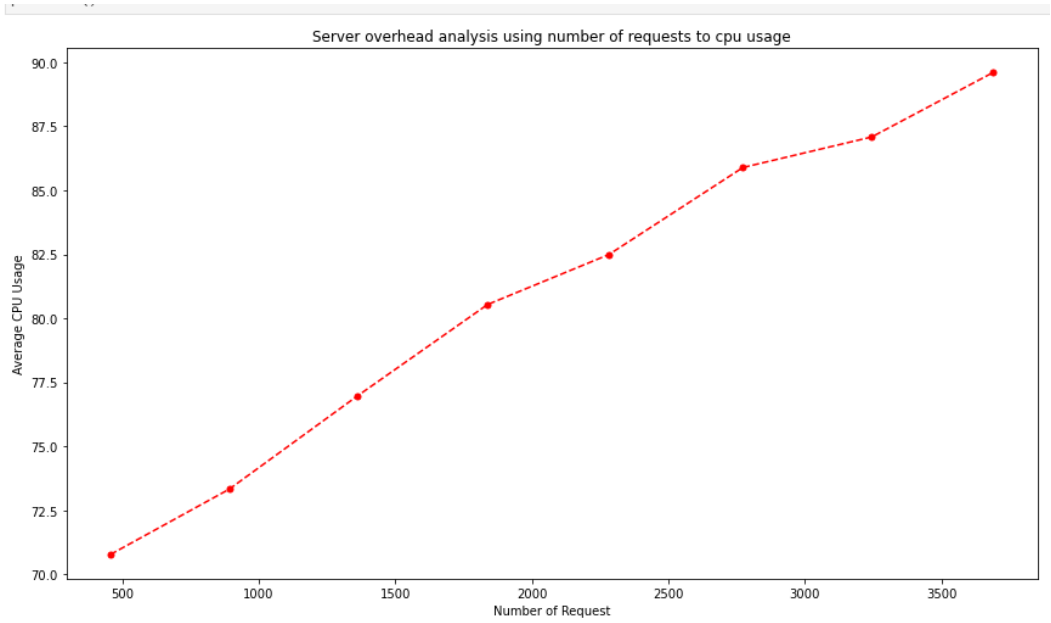

Figure 4: Performance of CGGD with respect to CPU usage on EUA dataset.

6.3 Discussion

In the experiment we conducted on optimization of edge server selection technique, we implemented model that combines GA and GD (CGGD). In the results demonstrated above, we find that heuristic ML approaches perform efficiently in reducing latency and maximizes computational resources significantly. While we obtain super results compared to the baseline algorithms, several factors may affect our results. These factors include but are not limited to the randomisation effect of user distribution, input data upload and download rate. The challenges to the validity of our work's applicability include whether our technique can be applied to other real-world scenarios in mobile edge computing. There is currently no dataset that has user distribution, user mobility path, and edge server distribution all at the same time. To minimize application risks, we examined CGGD over varying parameters such as user and base station distributions, and the proportion of output data size to input data size, in order to mimic as many application situations as feasible. Furthermore, we consider the randomness of different variables in each iteration of experiments to advance the achievability of CGGD in real life.

7 Conclusion and Future Work

Resource scarcity is a major hindrance that plagues mobile edge computing. The fact that mobile users are greedy makes solving the issue even the more difficult. The temporal and computational demands of mobile users is high and requires enough resources to fulfil user requests. A lot of research has been carried out on the optimization of selection of edge servers in mobile edge computing using machine learning. Given the large solution space, selecting the optimal server to run serve applications to mobile user is essential to reducing latency and overhead and improving the user’s quality of experience. Several optimization methods have been proposed using GA and other optimization techniques in edge server selection. Recent research shows the efficacy of heuristic methods in providing efficient ways in selecting the optimal edge server.

In this study we conducted on edge sever selection optimization experiment by implementing a hybrid technique of using two widely used techniques - genetic algorithm and gradient descent algorithm. Firstly, we measured the probability of a task request getting completed. Given that we assume a normal distribution, we can obtain the average response time. To minimize latency and computation overhead, we propose CGGD algorithm that can select servers in advance for the user by providing good quality of service. The outcome demonstrates that CGGD algorithm performs better than traditional algorithms both in terms of latency and CPU usage.

We propose the application of CGGD in dynamically offloading tasks to edge servers in the context of service migration for mobile users. By introducing other elements and specifications, we will aim to handle user requests at the edge of the network. In summary, this study has shown that using heuristic machine learning can evidently optimize edge server selection.

Acknowledgement

My Sincere appreciation goes to my supervisor **Sean Heeney**, for his guidance, support, patience and prompt response to my queries in the cause of project implementation. My heart appreciation and vote of thanks goes to my family Mrs. Pibowei Omamofe Taiwo (Mother), Mr. Peter Solari Pibowei(Father) and Mr. Oyindeinbofa Pibowei(Brother) for their support, encouragement and contributions to the actualization of this work, many thanks to **YOU**.

References

Yuan, B., Guo, S., & Wang, Q. (2021). Joint Service Placement and Request Routing in Mobile Edge Computing Networks. *2021 13th International Conference on Advanced Computational Intelligence, ICACI 2021*, 26–33. https://doi.org/10.1109/ICACI52617.2021.9435886

Thiruvasagam, P. K., Chakraborty, A., & Murthy, C. S. R. (2021). Latency-aware and Survivable Mapping of VNFs in 5G Network Edge Cloud. *2021 17th International Conference on the Design of Reliable Communication Networks, DRCN 2021*. https://doi.org/10.1109/DRCN51631.2021.9477372

Hadžić, I., Abe, Y., & Woithe, H. C. (2019). Server Placement and Selection for Edge Computing in the ePC. *IEEE Transactions on Services Computing*, *12*(5), 671–684. https://doi.org/10.1109/TSC.2018.2850327

Zhao, H., Deng, S., Zhang, C., Du, W., He, Q., & Yin, J. (2019). A mobility-aware cross-edge computation offloading framework for partitionable applications. *Proceedings - 2019 IEEE International Conference on Web Services, ICWS 2019 - Part of the 2019 IEEE World Congress on Services*, 193–200. https://doi.org/10.1109/ICWS.2019.00041

Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., & Leung, K. K. (n.d.). *Dynamic Service Migration in Mobile Edge-Clouds*.

Guidara, I., Guermouche, N., Chaari, T., Tazi, S., & Jmaiel, M. (2014). Pruning based service selection approach under QoS and temporal constraints. *Proceedings - 2014 IEEE International Conference on Web Services, ICWS 2014*, 9–16. https://doi.org/10.1109/ICWS.2014.15

Tan, H., Han, Z., Li, X. Y., & Lau, F. C. M. (2017). Online job dispatching and scheduling in edge-clouds. *Proceedings - IEEE INFOCOM*. https://doi.org/10.1109/INFOCOM.2017.8057116

Charyyev, B., Arslan, E., & Gunes, M. H. (2020). Latency Comparison of Cloud Datacenters and Edge Servers. *2020 IEEE Global Communications Conference, GLOBECOM 2020 - Proceedings*, 0–5. https://doi.org/10.1109/GLOBECOM42002.2020.9322406

Lin, H., Xu, X., Zhao, J., & Wang, X. (2020). Dynamic service migration in ultra-dense multi-access edge computing network for high-mobility scenarios. *EURASIP Journal on Wireless Communications and Networking*, *2020*(1), 191. https://doi.org/10.1186/s13638-020-01805-2

Wei, H., Luo, H., & Sun, Y. (2020). Mobility-aware service caching in mobile edge computing for internet of things. *Sensors (Switzerland)*, *20*(3). https://doi.org/10.3390/s20030610

Deng, S., Huang, L., Taheri, J., & Zomaya, A. Y. (2015). Computation Offloading for Service Workflow in Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*, *26*(12), 3317–3329. https://doi.org/10.1109/TPDS.2014.2381640

Selection, N. L. N. (2016). *QoS Prediction for Web Service Recommendation with*. *26*(4), 611–632. https://doi.org/10.1142/S0218194016400040

Alchalabi, A. E., Shirmohammadi, S., Mohammed, S., Stoian, S., & Vijayasuganthan, K. (2021). Fair Server Selection in Edge Computing with Q-Value-Normalized Action-Suppressed Quadruple Q-Learning. *IEEE Transactions on Artificial Intelligence*, *PP*(0), 1–1. https://doi.org/10.1109/tai.2021.3105087

Wang, S., Xu, J., Zhang, N., & Liu, Y. (2018). A Survey on Service Migration in Mobile Edge Computing. *IEEE Access*, *6*(c), 23511–23528. https://doi.org/10.1109/ACCESS.2018.2828102

Xenakis, D., Passas, N., Merakos, L., & Verikoukis, C. (2013). *Mobility Management for Femtocells in LTE-Advanced : Key Aspects and Survey of Handover Decision Algorithms*. 1–28.

Chen, M., Hao, Y., Hu, L., Hossain, M. S., & Ghoneim, A. (2018). Edge-CoCaCo: Toward Joint Optimization of Computation, Caching, and Communication on Edge Cloud. *IEEE Wireless Communications*, *25*(3), 21–27. https://doi.org/10.1109/MWC.2018.1700308

Zou, G., Qin, Z., Deng, S., Li, K. C., Gan, Y., & Zhang, B. (2021). Towards the optimality of service instance selection in mobile edge computing. *Knowledge-Based Systems*, *217*, 106831. https://doi.org/10.1016/j.knosys.2021.106831

Dilanka, G., Viranga, L., Pamudith, R., Gamage, T. D., & Ranaweera, P. S. (2021). *A Novel Server Selection Strategy for Multi-access Edge Computing*. *December*.

Zhang, Y., Zhang, W., Peng, K., Yan, D., & Wu, Q. (2021). *A novel edge server selection method based on combined genetic algorithm and simulated annealing algorithm*. https://doi.org/10.1080/00051144.2020.1837499

Chen, W., Wang, D., & Li, K. (2019). Multi-User Multi-Task Computation Offloading in Green Mobile Edge Cloud Computing. *IEEE Transactions on Services Computing*, *12*(5), 726–738. https://doi.org/10.1109/TSC.2018.2826544

Scoca, V., Aral, A., Brandic, I., De Nicola, R., & Uriarte, R. B. (2018). Scheduling latency-sensitive applications in edge computing. *CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science*, *2018-Janua*(Closer 2018), 158–168. https://doi.org/10.5220/0006706201580168

Zhang, W., Zhang, Y., Wu, Q., & Peng, K. (2019). Mobility-enabled edge server selection for multi-user composite services. *Future Internet*, *11*(9), 1–17. https://doi.org/10.3390/fi11090184

Soltani, S., Martin, P., & Elgazzar, K. (2018). A hybrid approach to automatic IaaS service selection. *Journal of Cloud Computing*, *7*(1). https://doi.org/10.1186/s13677-018-0113-8

Wu, H., Deng, S., Li, W., Yin, J., Li, X., Feng, Z., & Zomaya, A. Y. (2019). Mobility-aware service selection in mobile edge computing systems. *Proceedings - 2019 IEEE International Conference on Web Services, ICWS 2019 - Part of the 2019 IEEE World Congress on Services*, 201–208. https://doi.org/10.1109/ICWS.2019.00042

Guidara, I., Guermouche, N., Chaari, T., Tazi, S., & Jmaiel, M. (2015). Heuristic Based Time-Aware Service Selection Approach. *Proceedings - 2015 IEEE International Conference on Web Services, ICWS 2015*, 65–72. https://doi.org/10.1109/ICWS.2015.19

Tianze, L., Muqing, W., Min, Z., & Wenxing, L. (2017). An Overhead-Optimizing Task Scheduling Strategy for Ad-hoc Based Mobile Edge Computing. *IEEE Access*, *5*(c), 5609–5622. https://doi.org/10.1109/ACCESS.2017.2678102

Zeng, F., Ren, Y., Deng, X., & Li, W. (2019). Cost-effective edge server placement in wireless metropolitan area networks. *Sensors (Switzerland)*, *19*(1), 1–21. https://doi.org/10.3390/s19010032

Kasi, S. K., Kasi, M. K., Ali, K., Raza, M., Afzal, H., Lasebae, A., Naeem, B., Islam, S. U., & Rodrigues, J. J. P. C. (2021). Heuristic Edge Server Placement in Industrial Internet of Things and Cellular Networks. *IEEE Internet of Things Journal*, *8*(13), 10308–10317. https://doi.org/10.1109/JIOT.2020.3041805

Qian, H., Zu, H., Cao, C., & Wang, Q. (2013). CSS: Facilitate the cloud service selection in IaaS platforms. *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, 347–354. https://doi.org/10.1109/CTS.2013.6567253

Tang, L., Tang, B., Zhang, L., Guo, F., & He, H. (2021). Joint optimization of network selection and task offloading for vehicular edge computing. *Journal of Cloud Computing*, *10*(1). https://doi.org/10.1186/s13677-021-00240-y

Chen, M. H., Liang, B., & Dong, M. (2017). Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point. *Proceedings - IEEE INFOCOM*. https://doi.org/10.1109/INFOCOM.2017.8057150

Shah-Mansouri, H., & Wong, V. W. S. (2018). Hierarchical fog-cloud computing for IoT systems: A computation offloading game. *IEEE Internet of Things Journal*, *5*(4), 3246–3257. https://doi.org/10.1109/JIOT.2018.2838022

Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., & Bennis, M. (2018). Performance Optimization in Mobile-Edge Computing via Deep Reinforcement Learning. *IEEE Vehicular Technology Conference*, *2018-Augus*, 1–6. https://doi.org/10.1109/VTCFall.2018.8690980

Lee, G., Saad, W., & Bennis, M. (2017). An online secretary framework for fog network formation with minimal latency. *IEEE International Conference on Communications*, 1–6. https://doi.org/10.1109/ICC.2017.7996574

Zhang, M., Ranjan, R., Haller, A., Georgakopoulos, D., & Strazdins, P. (2012). Investigating decision support techniques for automating Cloud service selection. *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, 759–764. https://doi.org/10.1109/CloudCom.2012.6427501

Lai, P., He, Q., Abdelrazek, M., Chen, F., Hosking, J., Grundy, J., & Yang, Y. (2018). Optimal edge user allocation in edge computing with variable sized vector bin packing. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11236 LNCS*, 230–245. https://doi.org/10.1007/978-3-030-03596-9_15